Technology Transfer Issues for Formal Methods of Software Specification

Ken Abernethy
Department of Computer Science
Furman University
Greenville, SC 29613
Ken.Abernethy@furman.edu

John Kelly
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
John.Kelly@jpl.nasa.gov

Ann Sobel and James D. Kiper
Department of Systems Analysis
Miami University
Oxford, OH 45056
{sobelae,kiperjd}@muohio.edu

John Powell
NASA Ames Software IV&V Facility
100 University Drive
Fairmont, WV 26554
jpowell29@hotmail.com

Abstract

Accurate and complete requirements specifications are crucial for the design and implementation of high-quality software. Unfortunately, the articulation and verification of software system requirements remains one of the most difficult and error-prone tasks in the software development lifecycle. The use of formal methods, based on mathematical logic and discrete mathematics, holds promise for improving the reliability of requirements articulation and modeling. However, formal modeling and reasoning about requirements has not typically been a part of the software analyst's education and training, and because the learning curve for the use of these methods is nontrivial, adoption of formal methods has proceeded slowly. As a consequence, technology transfer is a significant issue in the use of formal methods. In this paper, several efforts undertaken at NASA aimed at increasing the accessibility of formal methods are described. These include the production of the following: two NASA guidebooks on the concepts and applications of formal methods, a body of case studies in the application of formal methods to the specification of requirements for actual NASA projects, and course materials for a professional development course introducing formal methods and their application to the analysis and design of software-intensive systems. In addition, efforts undertaken at two universities to integrate instruction on formal methods based on these NASA materials into the computer science and software engineering curricula are described.

1. Introduction and Background

One of the most challenging tasks in software development is to assure reliability of systems being designed and constructed. This becomes even more important as the use of software increases dramatically in embedded systems within life-critical environments such as medicine, air traffic control and other transportation systems, spacecraft control, and national defense weapons deployment and activation. A crucial step in gaining assurance in a system's reliability is to obtain a high degree of confidence that the system's requirements have been accurately captured and articulated.

Accurate and complete requirements specifications are crucial because a system's design and implementation derive their quality directly from the quality of the requirements specification. Several studies point to problems arising in the requirements specification phase of the software life cycle as an important factor in the failure to achieve sufficient quality levels. For example, it was found [13] that most software safety errors exposed during system integration and testing of two NASA spacecraft were traceable to defects in requirements and interface specifications. In a separate study, it was discovered [10] that the density of major defects found through the use of formal inspection techniques was seven times higher during the requirements phase than during the implementation phase. Clearly better requirements analysis methods are an important challenge in our attempts to increase software quality. Further, it has been observed that organizations using existing techniques are experiencing "quality ceilings," meaning that the traditional techniques have been fine-tuned to the point where major quality improvements can no longer be achieved through these methods – even though some defects still remain in the developed products [10].

Recent research is demonstrating the clear advantages of a more formal and mathematical approach to software requirements capture and design. Methods used in such an approach are collectively called formal methods for software specification, and these methods have been shown to provide added reliability by modeling requirements in a way that they can then be reasoned about in a rigorous and repeatable manner [8,19]. In general, the term *formal methods* refers to the use of techniques employing formal logic and discrete mathematics in the specification, design, and implementation of software (and hardware) systems.

While the application of formal methods to all phases of the software life cycle has proven productive, our interest in this paper is limited to the use of such methods in the requirements specification phase only. Formal modeling and reasoning about requirements has not been a part of the typical software analyst's education and training, and because the learning curve for the use of these methods is nontrivial, adoption of formal methods has proceeded slowly. As a consequence, technology transfer is one of the significant issues in the use of formal methods. We examine efforts at NASA and at several universities to facilitate this technology transfer.

2. Formal Methods Technology Transfer at NASA

It should be noted that there are many issues to be taken into account when considering incorporating formal methods in software development. There are many different levels of formality or rigor that can be adopted and a variety of formal methods can be used to address many different subtasks in a software development project. The cost/benefits analysis of the use of formal methods is in fact an involved task, with administrative and project management factors as well as technical factors to be considered. In addition, there are a great variety of tools and techniques that can be applied to the software development life cycle. These tools have wide ranges in the level of rigor they impose, their difficulty to learn and use, their cost, and the advantages and benefits they provide.

We describe here an effort undertaken at NASA to address these issues. This effort has resulted in two published Formal Methods Guidebooks and a Formal Methods Coursebook, all published by the NASA Office of Safety and Mission Assurance. This work has been supported by the NASA Software Program (UPN323-08), an agency wide program, managed by the NASA Software IV&V Facility, to promote continual improvement of software engineering within NASA.

2.1 NASA Guidebooks

The NASA Guidebook I [16] considers technical and administrative issues that must be addressed when establishing the use of formal methods on a project. The goal is to aid decision-makers in the successful application of formal methods to the development of high-quality systems at reasonable cost. The focus is on administrative and planning considerations for the successful application of formal methods.

More particularly, this volume of the guidebook is:

- written for project decision makers, including managers, engineers, and assurance personnel, who are considering the use of formal methods on their project;
- designed as an easily understood overview of important issues associated with the use of formal specifications; and
- intended to be a useful guide to planning and implementing formal methods on a project.

Modifications to release 2.0 include updated tools descriptions, added case study information, modification to the levels of rigor for applying formal methods, new references, and many other improvements.

The NASA Guidebook II [17] presents technical issues involved in applying formal methods to specify and analytically verify software systems. Practical techniques and strategies for verifying requirements and high-level designs for software intensive systems are discussed. The focus is on illustrating the growing practicality of applying formal methods for enhancing the quality of aerospace and avionics applications. This work is also a product of the NASA Software Program and was cooperatively developed by a three-center NASA team from the Jet Propulsion Laboratory, Johnson Space Center, and Langley Research Center.

More particularly, volume II of the guidebook is designed to:

- help the transition of formal methods from experimental use into practical application for critical software requirements and design within NASA;
- provide guidance for the novice practitioner; and
- discuss technical issues involved in applying these techniques to aerospace and avionics software systems.

Volume II also includes an illustrated example on a NASA application: the Simplified Aid for EVA Rescue (SAFER). SAFER is a system for free-floating astronaut propulsion that is intended for use on Space Shuttle missions, as well as during Space Station construction and operation. It is a small, self-contained, backpack propulsion system enabling free-flying mobility for a crewmember engaged in extravehicular activity (EVA) that has evolved as a streamlined version of NASA’s earlier Manned Maneuvering Unit (MMU).

Both guidebooks went through extensive review by representatives from NASA centers, formal methods experts, government, industry, and academia. They are being used as supplementary survey reading material in several college courses. Copies are available upon request from J. Kelly and electronically at http://eis.jpl.nasa.gov/quality/Formal_Methods/home.html.

2.2 Case Studies

The guidebooks reflect experience in pilot testing formal methods on NASA’s software subsystems. Prior to the production of the guidebooks, several case studies were conducted to explore the use of formal methods to verify requirements and design within NASA applications. These applications were drawn from the Space Shuttle, Space Station, Cassini (JPL) and SAFER projects. Guidebook I provides a short overview of each case study while Guidebook II provides the in-depth discussion of SAFER. Several papers [7,11,13,15] discuss the non-SAFER examples in detail.

Among the lessons learned from these NASA case studies are:

- The learning curve for formal methods is not prohibitive.
- Requirements analysts were willing and able to understand formal specifications with little training (reading versus writing formal specifications).
- The formal specification languages and tools used in the case studies were sufficiently mature for analyzing software requirements.
- The projects demonstrated the feasibility and usefulness of higher levels of abstraction for stating requirements.
- There is a need to develop libraries of formal specifications to support large and complex software designs.
- Proofs of safety properties helped to assure the absence of unsafe states in requirements and design.
- Rigorous, unambiguous basis for analyzing software requirements and design is not addressed elsewhere.
- Formal methods are useful in exposing previously undocumented implicit information.
- For critical subsystems the costs are not prohibitively expensive.
- Case studies demonstrated that an outside team could apply formal methods to a mature application and find significant issues.
- Formal methods practitioners should be willing to compromise and fill in the requirements analysis gaps with traditional techniques in addition to using formal specification languages.
- Selecting portions of the requirements of large space application for which formal methods provides the greatest analysis leverage is nontrivial.
- Formal methods need to be integrated with other validation and verification (V&V) techniques (e.g. Fagan inspections, traceability analysis, hazards analysis, etc.).
- Formal methods should be introduced on projects that already have in place good solid V&V procedures. They are not a replacement for doing basic V&V procedures.
- Formal methods uncovered significant issues in mature requirements.
- Even in an unstable requirements environment, formal specifications were found to be beneficial.
- Requirements analysts and developers were generally impressed by the thoroughness and insight of the issues lists produced by the pilot study team.
- Object Modeling Technique provided a good road map before developing formal specifications.
- The formal methods tools were more robust and mature than originally expected.

2.3 Formal Methods Professional Development Course

Over the past several years, an effort coordinated by JPL has resulted in the production of course materials [3] to support a three-day professional development course introducing software developers to the concepts and applications of formal methods for software specification. This course emphasizes the use of formal methods for the verification of software requirements and design. Formal methods for software specification depend on explicit descriptions of assumptions, constraints, and characteristics of a system toward the goal of reducing reliance on intuition and judgment in evaluating the consistency and correctness of a system's specification. A primary focus of the course is on basic FM/AV techniques and strategies. The specification languages Z, SPIN, and PVS [4,5,18,21] are introduced as representative and extended examples are developed with each. This course complements material found in the NASA Formal Methods Guidebook, Volumes I and II.

3. Use of the FM Course Materials with NASA Contractors

The NASA formal methods course has been offered at JPL, Lewis/Glenn Research Center, and the NASA Ames Independent Verification and Validation (IV&V) Facility in Fairmont, WV and was attended by a variety of personnel including NASA employees as well as contractor employees. The course was given in ten modules over three days and was team-taught by K. Abernethy, J. Kelly, and J. Powell. The ten modules covered were:

1. An Overview of Formal Methods for Software Analysis
2. General Formal Methods Concepts
3. Modeling Requirements using the Language Z
4. An Overview of Mathematical Models
5. Modeling Requirement using Model Checking
6. Formal Specification Languages and Analysis
7. Modeling Requirements using PVS
8. A Background on Formal Proof Analysis
9. A Case Study of Requirement Model using PVS
10. Issues in Integrating Formal Methods on a Project

The potential benefit of such a course offering is illustrated by the experience at the NASA IV&V Facility. Several employees of a NASA contractor attended this course offering. Although these class participants had little or no prior experience with formal methods, two members made use of the model checking instruction on several actual projects that were under way.

One participant employed model checking on proposed design for Redundancy Management of platforms and subsystems on the CLCS (Checkout and Launch Control System for the Space Shuttle) project. It was modeled using SPIN. Simulation of the message-passing scheme, including dropped messages, was also attempted. The initial 'prototype' model may be used as the basis for modeling the future Redundancy Management Rules Engine that will handle Fault Isolation, Identification and Corrective Action.

Another member of the class teamed with a domain expert to make use of model checking to verify anomalies in the MAGR (Miniature Airborne GPS Receiver) on board the Space Shuttle. Their use on the model checker SPIN was somewhat novel. Because they already knew of suspicious paths through the system that would potentially cause problem, were under deadline pressure, and were using SPIN for the first time, they built their model using Pearl scripts explicitly to be used in the interactive simulation mode. Through follow-up dialogues with J. Callahan (of the NASA IV&V Facility) and J. Powell, they were able to temporarily forgo the use of Linear Temporal Logic (LTL) properties. This allowed them to successfully make use of the model checking technology very rapidly at a point well into the project.

Subsequently, the successful use of SPIN on the MAGR project resulted in a request that J. Powell present an overview talk about SPIN's capabilities in full to the MAGR project members and management. The purpose of the talk was twofold. The first was to demonstrate the success gained by use of the SPIN tool in its limited use on the MAGR project; the second was to stress the full capability of SPIN, including LTL property verification for use on future projects. Subsequent feedback after the talk was very positive in regard to future use of SPIN on upcoming projects.

4. Incorporation of FM Concepts in the CS Curriculum

The recognized importance of formal methods and the scarcity of professional software developers trained in their use presents both an opportunity and a challenge for computer science educators. While there is growing recognition in a number of organizations of the potential value of using formal specification methods, there is an attendant reluctance to commit the resources needed to implement these methods. One of the major considerations in implementing these methods in most organizations is the lack of software developers with the necessary background and expertise in formal methods. Further, the learning curve for these methods is reasonably steep (especially in a self-learning mode), hence deterring efforts to initiate on-the-job training. Formal methods involve a high degree of mathematical formalism, and hence require a corresponding degree of commitment on the part of the learner to achieve a level of comfort approaching that most software developers have with traditional requirements analysis methods, with their dependence on English-like specifications. Generally, organized workshops and training courses are required to move a software development team to the point of applying formal methods to actual projects.

While in-service workshops and technical courses are necessary to move organizations into the use of formal specification methods, these efforts alone will not be sufficient. Once expertise is gained, it must be passed along to new developers as they are hired. To accomplish this, the whole process would need to be repeated, because the level of rigor and formality involved would make an informal transfer of the necessary knowledge difficult at best. While many organizations may be willing to invest in training and education to initiate the use of formal methods, most organizations are not anxious or even able to assume the long-term obligation of an on-going in-house educational system. Rather, this burden should properly be shifted to colleges and universities producing the next generation of software developers.

In this section, we will explore several efforts currently underway to incorporate some of the NASA materials described earlier into university computer science curricula. Three approaches are described:

- A spiral method in which formal methods are incorporated in existing CS courses in brief modules. The level of rigor and depth of these modules increases as the student progresses through the curriculum.
- A complete revision of the undergraduate curriculum to incorporate formal methods. This revision includes introduction of new courses and modifications to some existing ones.
- Introduction of a single course to present formal methods at the end of the curriculum.

4.1 Formal Methods at Furman University

How can the introduction of formal methods be incorporated into already tight computer science and software engineering curricula? An approach being explored at Furman University is to introduce brief modules, as opposed to entire courses, integrated over several traditional computer science courses. The formality of the methods being introduced makes it difficult for students to absorb and internalize them in one exposure, and so a spiral approach would seem desirable.

There are a number of feasible opportunities for integrating a spiral introduction to formal methods into the computer science curriculum [1]. Beginning modestly, we could incorporate a total of three to four weeks of exposure to these methods, while enhancing our current curriculum at the same time. The natural starting place is in the discrete structures course, where some introduction to formal logic is usually included already. The use of the specification language Z, for example, provides an outstanding practical application of formal logic that is often perceived by students as a very esoteric subject. Thus, we can accomplish a brief introduction to formal specification methods (with a corresponding discussion of the software life cycle), and at the same time, provide a meaningful application of formal logic in a computer science context [6]. In our opinion, a meaningful application of formal logic is far superior to the forced and contrived examples most textbooks currently use.

Of course, a spiral approach must include a second (and perhaps additional) exposure to material introduced earlier. A second group of courses for which the inclusion of perhaps a more extensive module on formal specification methods would be most appropriate is the software engineering and systems analysis courses. Here, issues of quality assurance are central and methods for the specification of software requirements are explored in some depth. The use of several examples of specifications developed in Z and another specification language, like PVS, would be a natural way to introduce students to the important topic of formal methods and has proven very successful at Furman [2]. In addition to the value of learning about these methods, the reason for exploring formal methods (i.e. the "quality ceiling" question inherent in traditional methods) makes an excellent springboard for a general discussion of how the demands for higher quality and more reliable software are driving the evolution of software engineering methodologies.

4.2 Formal Methods at Miami University

At Miami University, several faculty members have become aware that it is becoming imperative that we give our students training in the use of formal methods. This training has been incorporated in several meaningful ways. The most dramatic is in an NSF-funded project to incorporate formal methods throughout the undergraduate curriculum. Another is a more modest changed to a particular course – Advanced Software Engineering. This former, more extensive effort is described in detail at the web site http://www.sas.muohio.edu/san/formal/ and in a publication [20]. We will summarize this project here.

This modification to the curriculum of the Systems Analysis (SAN) Department of Miami University was an experimental course track that integrates formal methods into the core curriculum. It consisted of six courses, listed chronologically below. Two of the courses (Introduction to Program Derivation and Formal Analysis of Concurrent Programs) were new courses in the department, and dealt solely with formal analysis. The four others were existing courses, but were modified to include material on formal analysis. The courses can be characterized by the following titles:

- Introduction to Program Derivation
- Semantics of Data Structures
- Object-Oriented Design
- Formal Analysis of Concurrent Programs
- Software Engineering
- Practicum in Software Development

Since the conclusion of the second course, the goals of this experimental curriculum have been evaluated in both qualitative and quantitative ways. "Direct comparisons of the problem solving skills of the formal methods and control student groups have shown that the formal methods students possess an increased level of complex problem solving skills as well as a greater ability to perform problem abstraction." [20]

As described above, another approach to incorporating formal methods into a curriculum is through the use of a separate course at the end of the curriculum. Such a course has evolved as a separate effort from the more encompassing one described previously. A master's level course entitled Advanced Software Engineering and taught by either A. Sobel or J. Kiper has evolved to focus primarily on formal software engineering methods. Although this is a master's level course, this master's degree is a "retraining" program in which students with a Bachelor's degree in some other field are retooled in software engineering. Thus, the level of student knowledge is comparable to that of upper-division undergraduates.

This course has evolved over the years to have a large formal methods component. In the last instantiation (summer 1999), the course had three primary topics: Personal Software Process [9], the Cleanroom software development process [23], and the formal specification language Z [5,22]. The later two topics are in the domain of formal methods. The class learned about the more formal aspects of the Cleanroom method including the use of functional specifications in writing programs. Students also learned to use the formal specification language Z to write specifications of program segments. They used the tool Z Formaliser [24] to help learn the syntax. Z Formaliser is a syntax-directed editor that produces syntactically correct specifications. From prior experience, it has been learned that the use of such a tool is vital to student mastery of the language syntax. This mastery is, of course, the first step in understanding language semantics.

5. Conclusion

Clearly better requirements analysis methods remain an important challenge as we seek to advance software quality assurance. The two NASA guidebooks described here are designed to introduce both project managers and software developers to basic formal methods concepts and some of the most important issues involved in deciding when and where the application of formal methods is advisable and cost effective. The described professional development course materials complement and extend the guidebooks in focused areas. Particularly, these materials provide guided explorations of three representative formal methods tools for the modeling of requirements: Z, PVS, and SPIN. Our experience in offering three-day courses based on these materials has been very positive. Equipped with this introduction to formal methods and some follow-up consultations, practicing software developers have successfully applied formal methods to requirements specification on real projects. Further, these course materials and guidebooks have proven to be good resources for instructors who desire to integrate formal methods into university computer science and software engineering curricula. We believe that the combination of professional development workshops for managers and practicing software developers and enhancements in the concepts and use of formal methods to university courses provide two constructive approaches to raising the awareness in the software development community of some promising new techniques and methods for requirements modeling.

6. Bibliography

[1] Abernethy, K., “Formal methods for software specification: Implications for the undergraduate computer science curriculum,” *The Journal of Computing in Small Colleges*, 12: 3(20-29), 1996.

[2] Abernethy, K., M. Fletcher, A. Henderson, A. Hoffman, and K. Treu, “Formal methods for software specification: a case study in PVS,” *The Journal of Computing in Small Colleges*, 13: 3(22-30), 1997.

[3] Abernethy K., J. Kelly, and J. Powell, *Introducing Formal Methods*, Version 1.2, Office of Safety and Mission Assurance, National Aeronautics and Space Administration, Washington, DC, 1998.

[4] Butler, R., *An Elementary Tutorial on Formal Specification and Verification Using PVS*, NASA Technical Memorandum Number 108991, June 1993.

[5] Diller, A., *Z: An Introduction to Formal Methods*, John Wiley and Sons, New York, 1990.

[6] Grassmann, W. K. and J. Tremblay, *Logic and Discrete Mathematics: A Computer Science Perspective (Chapter 8)*, Prentice Hall, Inc., Upper Saddle River, NJ, 1996.

[7] Hamilton, D., R. Covington, and J. Kelly, “Experiences in applying formal methods analysis of software and system requirements,” *Proceedings of the Workshop on Industrial-Strength Formal Specifications Techniques* (WIFT ’95), Boca Raton, FL, April 5-8, 1995.

[8] Holloway, M (editor), *Proceedings of the Third Annual NASA Langley Formal Methods Workshop*, NASA Langley Research Center, Hampton, Virginia, June, 1995.

[9] Humphrey, Watts S., *Introduction to the Personal Software Process*, Addison-Wesley, 1997.

[10] Kelly, J., J. Sherif and J. Hops, “An analysis of defect densities found during software inspections,” *Journal of Systems and Software*,17(111-117), January 1992.

[11] Kelly, J., R. Covington, and D. Hamilton, “Results of a formal methods demonstration project,” *Proceedings, WESCON/94 & Idea/Microelectronics Conference*, Los Angeles, CA, Sept. 27-29, 1994.

[12] Kelly, J., K. Abernethy and J. Powell, “Start-up materials for software formal methods/analytical verification on your next project,” 2nd Annual Assurance Technology Conference, NASA Glenn Research Center, Cleveland, Ohio, May 26-7, 1999.

[13] Lutz, R., “Analyzing software requirements errors in safety critical embedded systems,” *Proceedings of IEEE International Symposium on Requirements Engineering*, pp. 126-133, San Diego, CA, January, 1993.

[14] Lutz, R. and Y. Ampo, “Experience report: using formal methods for requirements analysis of critical spacecraft software,” *Proceedings of the Nineteenth Annual Software Engineering Workshop*, NASAS Goddard Space Flight Center, Greenbelt, MD, December 1994.

[15] NASA, *Formal Methods Demonstration Project for Space Applications – Phase I Case Study: Space Shuttle Orbit DAP Jet Select*, JPL Document D-11432, December 22, 1993.

[16] NASA, *Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Vol. I: Planning and Technology Ins*ertion, [NASA/TP-98-208193], Release 2.0, National Aeronautics and Space Administration, Washington, DC, 1998, 88 pages.

[17] NASA, *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion*, [NASA-GB-001-97], Release 1.0, National Aeronautics and Space Administration, Washington, DC, 1997, 239 pages.

[18] Owre, S., N. Shankar, and J. Rushby, *User Guide For The PVS Specification And Verification System, Three Volumes: Language, System, and Prover Reference Manuals*, Computer Science Laboratory, SRI, International, Menlo Park, CA, 1995.

[19] Rushby, J., *Formal Methods and Digital Systems Validation for Airborne Systems*, NASA Contractor Report 4551, December 1993.

[20] Sobel, A., “Final results of incorporating an operational formal method into a software engineering curriculum,” *Proceedings of the IEEE Frontiers in Education Conference*, November 1999.

[21] Spivey, J. M. *Understanding Z, A Specification Language and its Formal Semantics*, Cambridge University Press, Cambridge, 1988.

[22] Spivey, J. M. *The Z Notation: A Reference Manual*, Prentice Hall, Inc., Upper Saddle River, NJ, 1989.

[23] Stavely, Allan M., *Towards Zero-Defect Programming*, Addison-Wesley, 1999.

[24] *Z Specific Formaliser User Guide*, version 7.5.6e Logica UK Ltd, April 1999.